7. LISTÁK

Ezen adatszerkezetnek számos változata van, melyeket az alábbi 3 fajta tulajdonság különbözteti meg egymástól ???:

- Egyirányú vagy kétirányú,
- Egyszerű vagy ciklikus,
- Fejelemes vagy fejelem nélküli.

Ezek közül azonban a jegyzetben csak az egyirányú egyszerű lista fejelemes ill. fejelem nélküli változatával foglalkozunk bővebben.

7.a.) Egyirányú, egyszerű lista

7.a.) 1. ADT

A lista ADT szintjét csak intuitív szinten tárgyaljuk (???).

Műveletek:

Üres:	$\rightarrow L$	Üres lista konstans; az üres lista „létrehozása”.
Üres?:	$L \rightarrow \mathsf{L}$	A lista üres voltának lekérdezése.
Elsőre:	$L \rightarrow L$	Az aktuális elem-komponens első elemre állítása.
Következőre:	$L \rightarrow L$	Az aktuális elem-komponens következő elemre állítása.
Érték:	$L \rightarrow E$	Az aktuális elem értékének lekérdezése.
Módosít:	$L \times E \rightarrow L$	Az aktuális elem értékének módosítása.
Töröl:	$L \rightarrow L$	Az aktuális elem törlése.
BeszúrUtán:	$L \times E \rightarrow L$	Adott értékű elem beszúrása az aktuális elem után.
BeszúrElsőnek:	$L \times E \rightarrow L$	Adott értékű elem beszúrása első elemként.
Utolsó?:	$L \rightarrow \mathsf{L}$	Annak lekérdezése, hogy az aktuális elem az utolsó-e.

7.a.) 2. ADS

Az egyirányú, egyszerű lista lineáris adatszerkezet; az alapvető szerkezetet ábrázoló gráf:

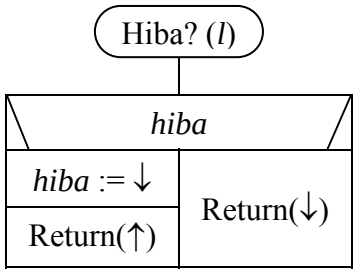
7.a.) 3. Reprezentációs szint

- Láncolt ábrázolás:

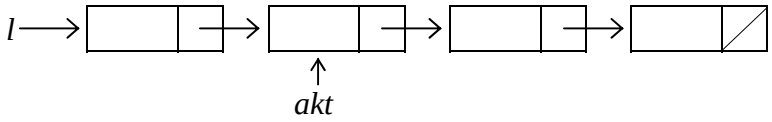
A listát rekordszerkezetként valósítjuk meg, melynek komponensei:

- A tulajdonképpeni lista pointere (*l* vagy *FEJ*),
- Az aktuális elemre mutató pointer (*akt*),
- Hibát jelző logikai változó (*hiba*),
- Esetleges további komponensek lehetnek: elemszám, előző, utolsó, stb..

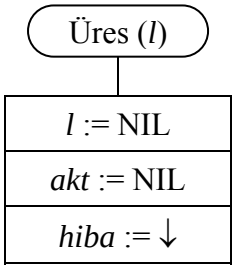
A *hiba* változó használatát az a filozófia magyarázza, mely szerint nem „bolondbiztos” programot kell írnunk: programozóként minden művelet esetén lekérdezzük (egy „Hiba?” nevű művelettel), hogy bekövetkezett-e hiba, majd a hibajelzés törlődik.??? Hibajelzést a „HIBA” nevű paranccsal idézhetünk elő, mely tulajdonképpen egy „*hiba* := ↑” utasítás.

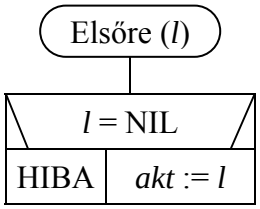


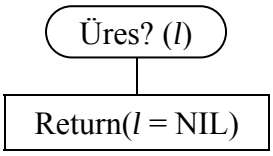
Fejelem nélküli lista:

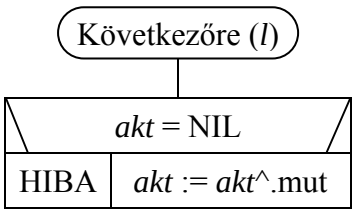


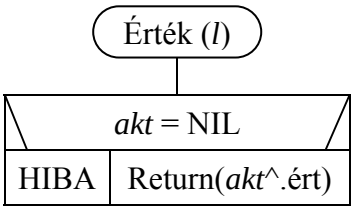
Az *akt* pointer értéke NIL lesz üres lista esetén, de akkor is, ha „lelépünk” a listáról.

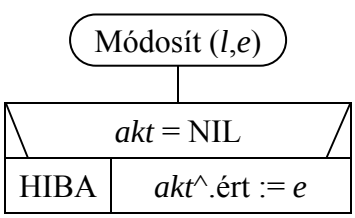


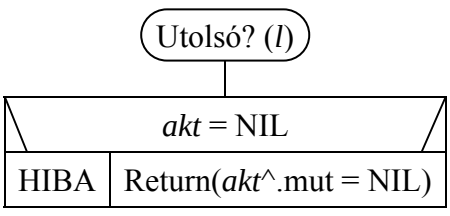


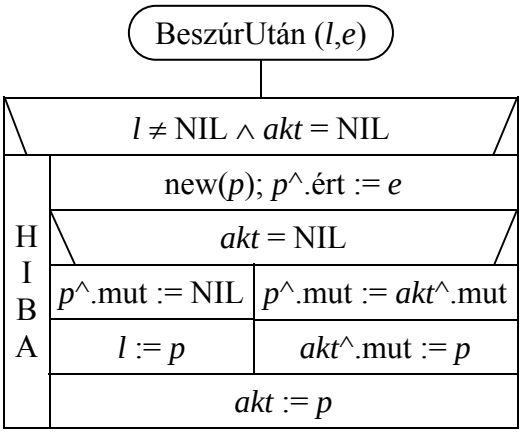


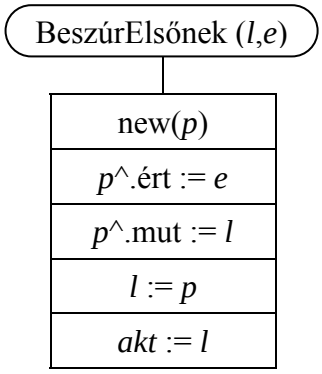


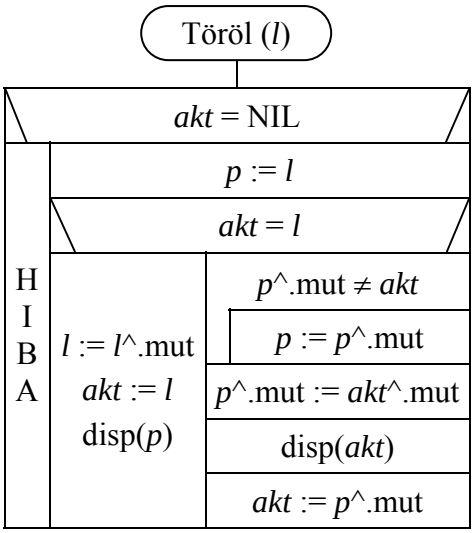




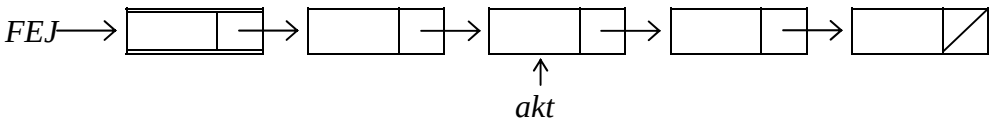




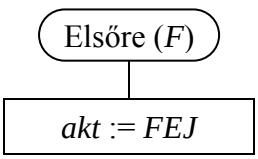


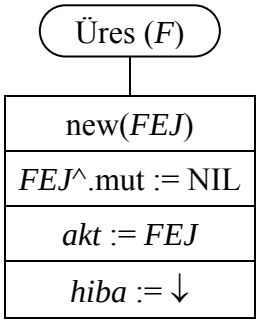


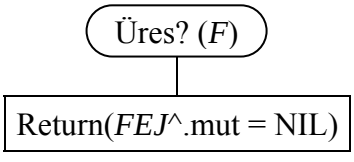
Fejelemes lista:

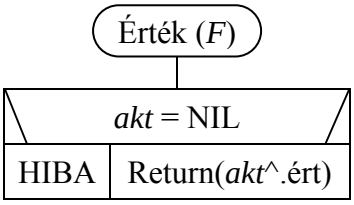


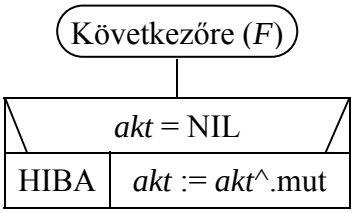
Ennél a megvalósításnál mindig létezik egy fejelem, melyben speciális adat van. Üres lista esetén a fejelem pointere NIL, az *akt* pointer a fejelemre mutat. Műveletek???

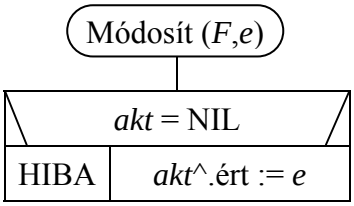


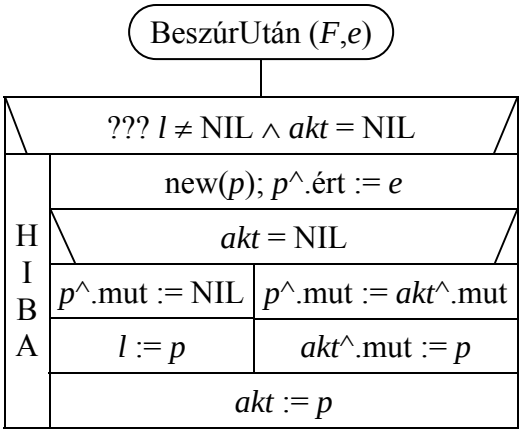


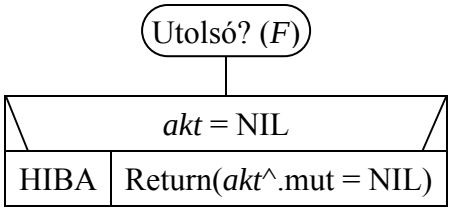


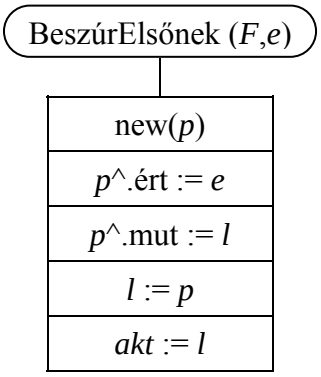


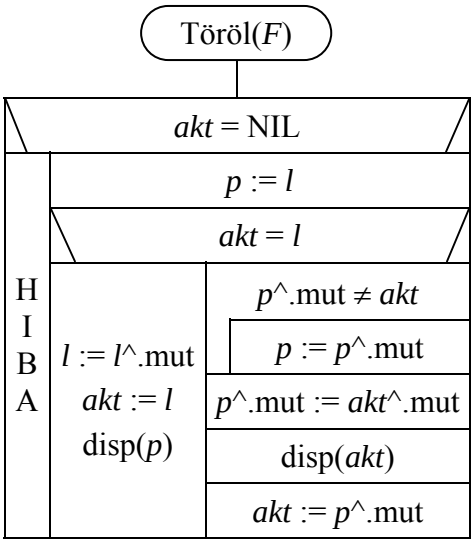








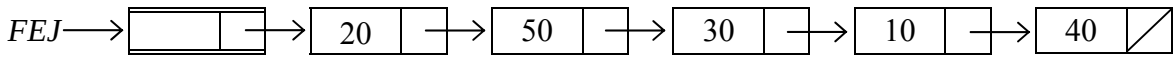




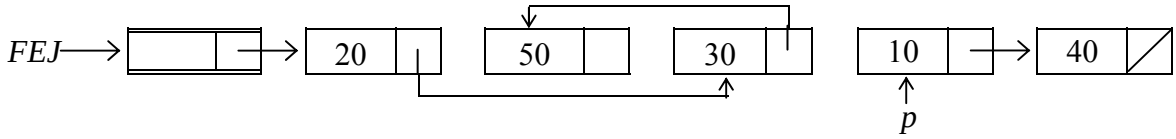
Példa a reprezentáció szintjén megoldott feladatra: A beszúró rendezés listás változata.

Pár szó erről a rendezésről ???

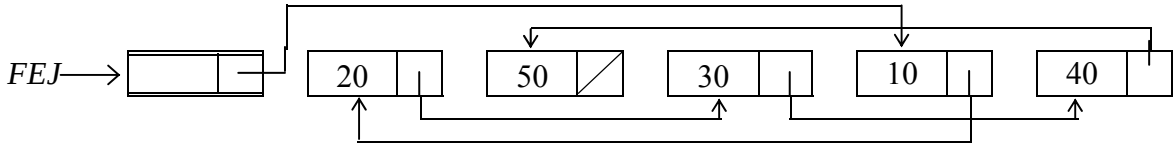
A rendezendő lista kezdetben:

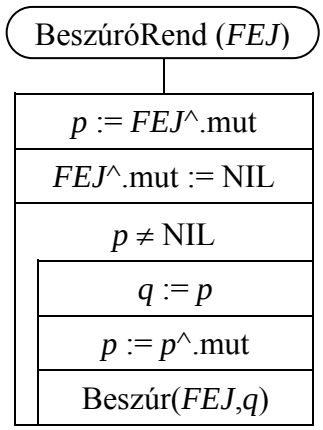


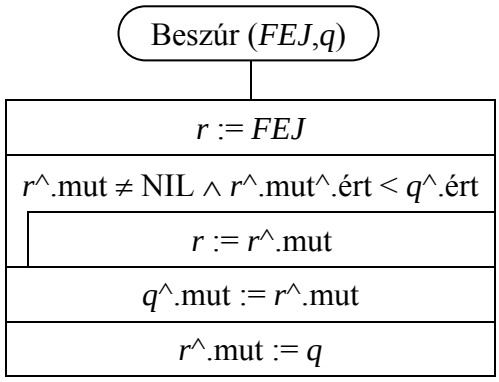
Egy közbülső állapot: következő lépésként a 10-es elemet leválasztjuk, és a már rendezett részbe, a megfelelő helyre láncoljuk.



Rendezés után:

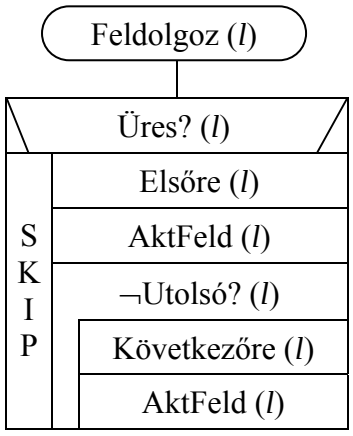






Az *r* pointer előre nyúlva keresi a beszúrandó elemnél nagyobb vagy egyenlő elemet, a ciklus éppen az előtt fog megállni.

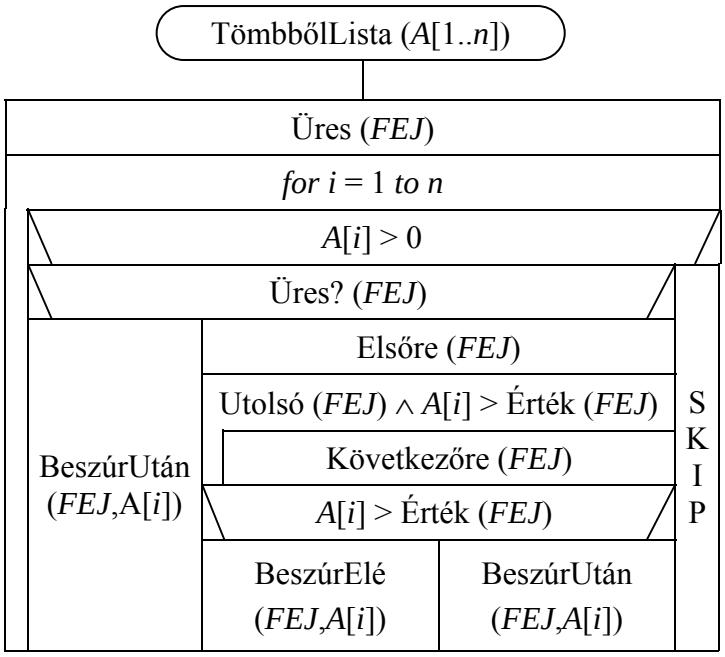
Műveletek szintjén megoldott feladatok:



Egy lista tipikus feldolgozási sémája

A fenti programban az AktFeld (*l*) egy közelebbről nem definiált tevékenység, amely az aktuális elem ismeretében elvégezhető.

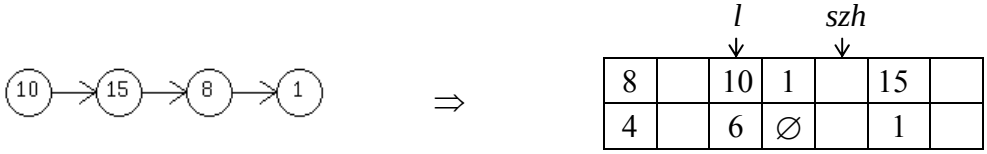
Példa: Adott tömb elemei közül a pozitívak helyezzük el rendezett módon egy fejelemes listába.



Példa a műveletek szintjén megoldott feladatra

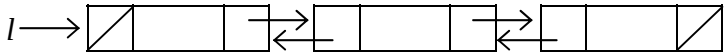
– Aritmetikai (statikus láncolt) ábrázolás:

Ezt az ábrázolási módot itt csak megemlítjük, mert ???

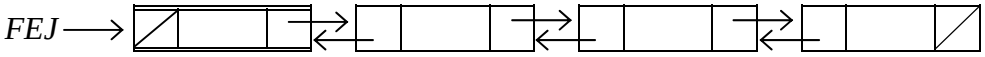


7.b.) Kétirányú, egyszerű lista

Fejelem nélküli lista:



Fejelemes lista:



7.c.) Ciklikus lista

Példa: Kétirányú, fejelemes ciklikus lista